Chapter 11
Development Team

In this chapter I describe the development team role. I begin by discussing five principal responsibilities of this role and conclude by describing ten characteristics that a development team should exhibit.

Overview

Traditional software development approaches define various job types, such as architect, programmer, tester, database administrator, UI designer, and so on. Scrum defines the role of development team, which is simply a cross-functional collection of these types of people. In particular, the development team is one of the three roles on every Scrum team. The development team's members, collectively, have the skills required to deliver the business value requested by the product owner.

The term *development team* may appear to be the wrong label to apply to a team that is composed of more than just developers. Other labels have been used, such as *delivery team*, *design-build-test team*, and just *team*. It's not apparent that any of these labels is more appropriate, less ambiguous, or easier to use than *development team*. For now, the Scrum community has converged on the use of the term *development team*, and I will use that term in this book.

Role-Specific Teams

Many organizations are accustomed to intentionally splitting different job roles into specialized, role-specific teams. These organizations might have one team of designers, one of developers, and another of testers. These teams hand off work to one another when it is complete and more or less function independently of each other.

In Scrum, the development team must do all of the work to produce one or more vertical slices of working product functionality each sprint, including the design, development, integration, and testing of that functionality. Thus, we need a team that is skilled at all of those tasks.

Some organizations try to maintain a separate testing or QA team while doing Scrum. Now, I admit there are times when having a separate team that focuses specifically on testing might be necessary—for example, a regulatory requirement might be that a separate team perform a particular type of testing. However, most of the time there is no such need. Testing should be fully interwoven into the work that takes

place during every sprint. Therefore, the development team doing the work during that sprint should do the testing.

Whenever you can, you should create cross-functional teams. Parceling the work out to different role-specific teams is suspect and is likely a serious impediment to the successful use of Scrum. Make sure you have a real need (besides habit) for keeping any role-specific teams.

Principal Responsibilities

Figure 11.1 illustrates the Scrum activities, annotated with the principal development team responsibilities.

I will describe each of these responsibilities.

Perform Sprint Execution

During sprint execution, development team members perform the hands-on, creative work of designing, building, integrating, and testing product backlog items into increments of potentially shippable functionality. To do this, they self-organize

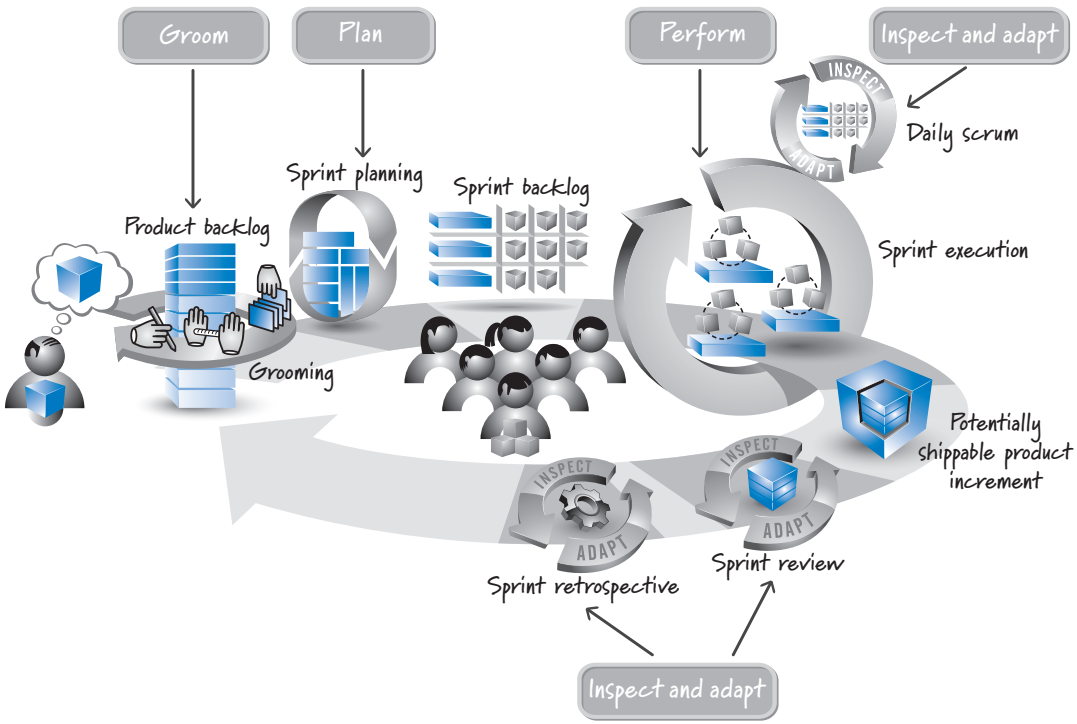

FIGURE 11.1 Development team responsibilities with respect to Scrum activities

and collectively decide how to plan, manage, carry out, and communicate work (see Chapter 20 for details). The development team spends a majority of its time performing sprint execution.

Inspect and Adapt Each Day

Each development team member is expected to participate in each daily scrum, during which the team members collectively inspect progress toward the sprint goal and adapt the plan for the current day's work. If some team members do not participate, the team can miss pieces of the big picture and may fail to achieve its sprint goal.

Groom the Product Backlog

Part of each sprint must be spent preparing for the next. A large part of that work focuses on product backlog grooming, which includes creating and refining, estimating, and prioritizing product backlog items (see Chapter 6 for details). The development team should allocate up to 10% of its available capacity every sprint to assist the product owner with these activities.

Plan the Sprint

At the beginning of each sprint, the development team participates in sprint planning. In collaboration with the product owner and with facilitation from the ScrumMaster, the development team helps to establish the goal for the next sprint. The team then determines which high-priority subset of product backlog items to build to achieve that goal (see Chapter 19). For a two-week sprint, sprint planning typically takes about half a day. A four-week sprint might need up to a full day for sprint planning.

Notice that planning happens iteratively. Rather than focusing on a very large, uncertain, and overly detailed plan at the start of a development effort, the team makes a series of smaller, more certain, and more detailed plans just in time at the beginning of each sprint.

Inspect and Adapt the Product and Process

At the end of each sprint, the development team participates in the two inspect-and-adapt activities: sprint review and sprint retrospective. The sprint review is where the development team, product owner, ScrumMaster, stakeholders, sponsors, customers, and interested members of other teams review the just-completed features of the current sprint and discuss how to best move forward (see Chapter 21). The sprint retrospective is where the Scrum team inspects and adapts its Scrum process and technical practices to improve how it uses Scrum to deliver business value (see Chapter 22).

Characteristics/Skills

Figure 11.2 illustrates important characteristics of the development team.

Self-Organizing

Team members self-organize to determine the best way to accomplish the sprint goal. There is no project manager or other manager who tells the team how to do its work (and a ScrumMaster should never presume to). **Self-organization** is a bottom-up, emergent property of the system—there is no external dominating force applying traditional top-down, command-and-control management.

Let me illustrate with an example. Where I live in Colorado, there is a pond at the entrance of my subdivision. Over the winter a flock of Canada geese comes and roosts there. So each year we have a couple of hundred geese that simultaneously make a big mess and are pretty to look at. Now, I also have two dogs named Letti and Toast. Normally they stay inside the fenced backyard. Occasionally we let them roam free outside the fence, and if they see the geese by the pond, they run down to greet them. I don't think they would hurt the geese, but when they see Letti and Toast coming, the geese decide to cede the pond to them for a while, so they take off en masse.

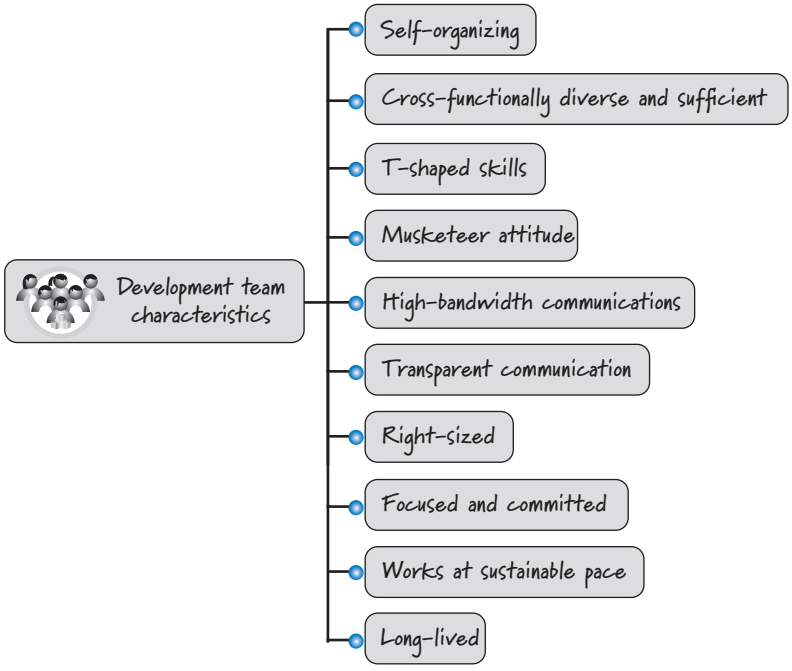

FIGURE 11.2 Development team characteristics

Did you ever wonder when birds take off how it is that they know to form their characteristic V pattern (flocking pattern)? Do you think there is a "manager bird," with a flipchart, down at my pond that calls a meeting to instruct the birds on how to flock (see Figure 11.3)?

I've lived by the pond for many years, and I don't ever recall seeing that meeting. (Although years ago my son Jonah declared, "Dad, you've never seen it because they do that meeting at night!" Hmm. Maybe he's on to something.)

No, unless my son is right and the birds are much craftier than I think, the geese flock through self-organization, a bottom-up emergent property of a **complex adaptive system**. In such systems, many entities interact with each other in various ways, and these interactions are governed by simple, localized rules operating in a context of constant feedback (see Figure 11.4).

These types of systems exhibit interesting characteristics, such as being remarkably robust and producing amazing novelty.

Like the flocking birds, a development team has no top-down command-and-control authority that tells the team how to do its work. Instead, a cross-functionally diverse team of people organize themselves in the most appropriate way to get work done. In effect, what emerges is the team's own equivalent of the V pattern.

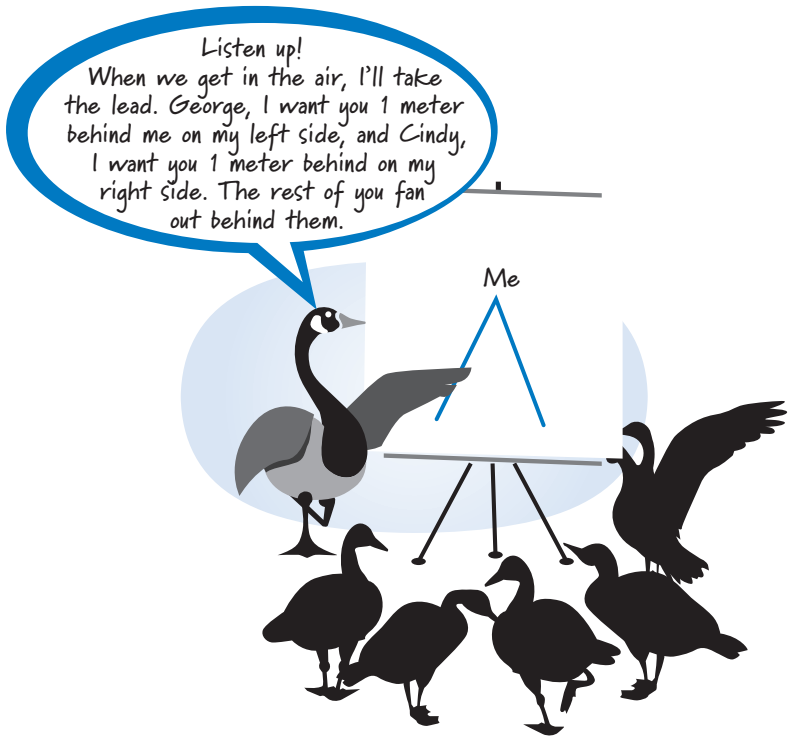

FIGURE 11.3 Flocking isn't the result of top-down planning.

FIGURE 11.4 Flocking: simple rules and frequent feedback

Managers, however, do have a vital role in Scrum. They create (and re-create) the environment for the self-organizing team. We'll talk more about the role of managers in Chapter 13.

Cross-Functionally Diverse and Sufficient

Development team members should be cross-functionally diverse; collectively they should possess the necessary and sufficient set of skills to get the job done. A well-formed team can take an item off of the product backlog and produce a good-quality, working feature that meets the Scrum team's definition of done.

Teams composed solely of people with the same skills (traditional silo teams) can at most do part of the job. As a result, silo teams end up handing off work products to other silo teams. For example, the development team hands the code off to the testing team, or the UI team hands off screen designs to the business logic team. Handoffs represent an excellent opportunity for miscommunication and costly mistakes. Having diverse teams minimizes the number of handoffs. And creating diverse teams doesn't prevent us from having multiple team members who might be highly skilled in the same discipline such as Java or C++ development or testing.

Cross-functionally diverse teams also bring multiple perspectives, leading to better outcomes (see Figure 11.5).

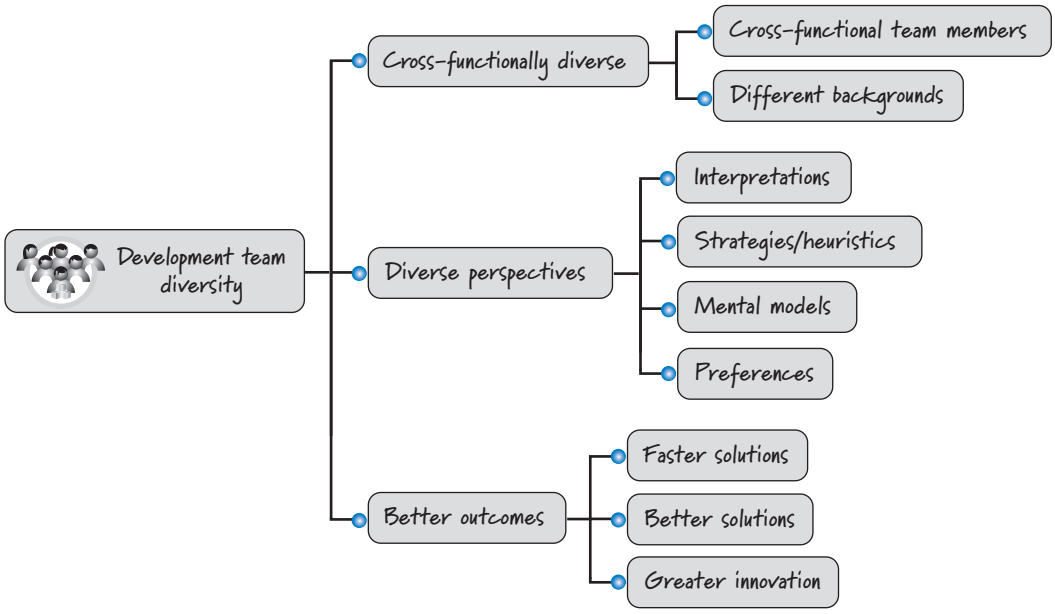

FIGURE 11.5 Team diversity

A cross-functionally diverse team has members from different backgrounds. Each team member brings a set of cognitive tools for problem solving; these tools can involve different interpretations (of the same data), different strategies (or heuristics) for solving problems, different mental models of how things work, and different preferences for both approaches and solutions. This kind of diversity typically leads to better outcomes in terms of faster solutions, higher-quality deliverables, and greater innovation, all of which translate into greater economic value (Page 2007).

We should also strive for team diversity by having a good mix of senior- and junior-level personnel on the same team. Too many senior-level people might cause unnecessary turbulence, similar to having too many cooks in the kitchen. Too many junior people, however, and the team might not be sufficiently skilled to get the job done. A good mix promotes a healthy, collaborative learning environment.

T-Shaped Skills

Flexible development teams are composed of members with T-shaped skills (see Figure 11.6).

T-shaped skills mean that a team member (say, Sue) has deep skills in her preferred functional area, discipline, or specialty. For example, Sue is a great user-experience (UX) designer—that is her specialty and where she prefers to do work. Sue,

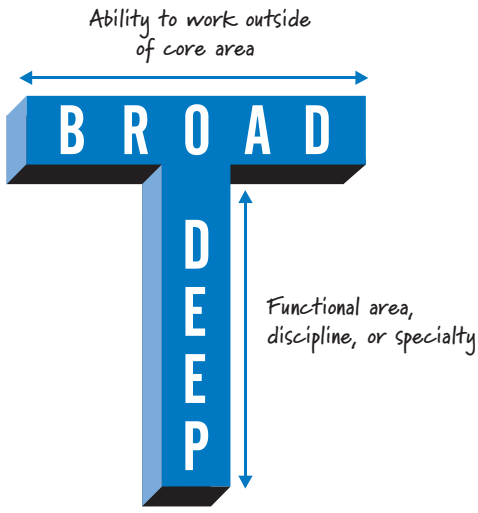

FIGURE 11.6 T-shaped skills

however, can also work outside of her core specialty area, doing some testing and some documentation. She isn't as good a tester or documenter as those who specialize in those areas, but she can help out with testing or documentation if that's where the team is experiencing a bottleneck and needs to swarm people to get the job done. In this respect Sue has broad skills that allow her to work outside her core area.

It's unrealistic to believe that every person on a team could work on every task. That's a lofty goal to have. For example, in domains with intense specialization, like video game development, where a team could have an artist, animator, audio engineer, artificial intelligence (AI) programmer, and tester, it's unreasonable to assume that everyone can do every job. If I were on a team developing a video game, I could work on the AI and do some testing, but I couldn't work on the art design (and you wouldn't want me to!). However, I might be able to help the artists with nonartistic design work such as using Photoshop to convert file formats or create scripts to apply operations on multiple files.

Managers should focus on forming teams that have the best set of T-shaped skills that are possible with available personnel. However, it might not be possible to get exactly the desired team skill set from the get-go, so the desired skill set could evolve over time as the needs of the product development effort evolve. Therefore, it is critical to have an environment where people are constantly learning and adding to their skill sets, whether those include domain knowledge, technical knowledge, thinking skills, or other capabilities. Management needs to support team members with time to learn and experiment (see Chapter 13).

Is it OK to have pure specialists on the team? Let's take our earlier example of Sue and assume she is a great UX designer, but that's all she can do. And, because we have

so few UX designers, we really don't want her doing anything but critical UX design work. We need her skills on the team, but we'll be able to fill only about 10% of her time with team-related work. In these cases, an obvious solution is to divide Sue's time among multiple teams.

However, we must be practical. Sue would be far too fractured if she divided her time in 10% increments to many teams at the same time. She would soon become a bottleneck (see the "Focused and Committed" section later in this chapter). Recall from Chapter 3 that our goal shouldn't be to keep people like Sue 100% utilized. Instead, we should be more concerned about the idle work (the baton sitting on the ground) that occurs when we rely too much on an overutilized resource. So, we might allocate Sue as a specialist to a reasonable number of products, but not so many that she is the cause of baton dropping.

Alternatively, because our goal is to achieve good flow with team members who have broad T-shaped skills, we should encourage Sue to help other team members acquire reasonable UX design knowledge so that we no longer need to rely so heavily on specialists.

To summarize, then, our goal is to form a team with members who have the proper skills to cover the core specialty areas and in aggregate have some overlap in skills to provide additional flexibility. To meet this goal, many team members should have T-shaped skills, but we still might have some specialists in the mix.

Musketeer Attitude

Members of the development team (and the Scrum team as a whole!) need to have the same attitude as the Three Musketeers—"All for one and one for all." This **Musketeer attitude** reinforces the point that the team members collectively own the responsibility of getting the job done. They win as a team or they fail as a team.

In a well-functioning Scrum team, I would never expect anyone to say, "I got my part done. You didn't get your part done. Therefore we failed." This attitude misses the point that team members are all in the same boat together (see Figure 11.7).

Team members must appreciate that they must work together to meet their commitments, because if they fail, it's going to be everybody's problem in the end. Having team members with a Musketeer attitude is critical to achieving shared success.

Having team members with T-shaped skills encourages this attitude and makes it practical because people are capable of working on more than one type of task. On these teams I don't expect to hear a person who is capable of doing the work say, "That's not my job."

However, because it is not always possible for a person to do every job, I might hear someone say, "I'm not capable of doing that job." In this case the team might choose to have the person without the skills apprentice with a person who has the skills so that in the future the team will have greater aggregate capabilities.

Even if skills limitations prevent people from working cross-functionally, team members can still organize their work to ensure a good flow through the sprint so

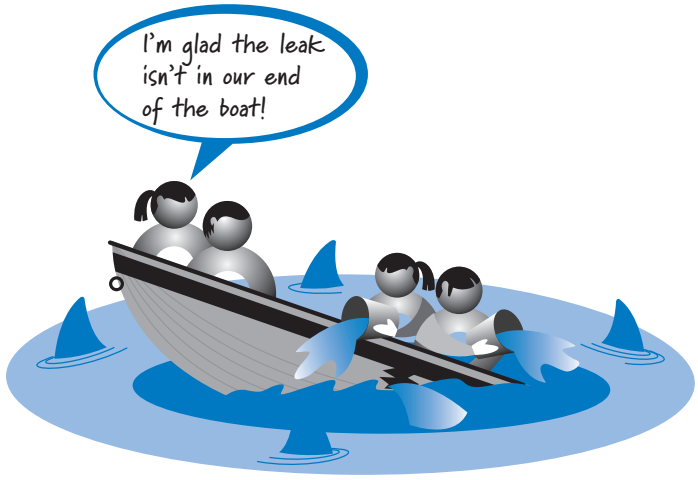

FIGURE 11.7 Team members must act as if they are all in the same boat.

that no one team member is overburdened. For example, holding all of the testing work until the end of the sprint so that the "tester" can do the work is most certainly a prescription for failure. See Chapter 20 for a deeper discussion of how the team should manage flow during sprint execution.

So, with a Musketeer attitude, no one is just "along for the ride." Each team member is responsible for making sure she is fully engaged at all times. Frequently this will mean speaking up and engaging in activities outside one's specialty to add to the diversity of the discussion. For example, although a team member's specialty might be testing, if she thinks there is a problem in the design the team is coming up with for a given feature, it's her duty to speak up, rather than shrug and say, "Not my job; they know better than I do anyway."

High-Bandwidth Communications

Development team members need to communicate with one another, as well as with the product owner and ScrumMaster, in a high-bandwidth manner, where valuable information is exchanged quickly and efficiently with minimal overhead.

High-bandwidth communications increase both the frequency and quality of information sharing. As a result, the Scrum team has more frequent opportunities to inspect and adapt, leading to better and faster decision making. Because the economic value of information is time-sensitive, accelerating the rate of information sharing allows the team to maximize its value. By quickly exploiting emergent opportunities and recognizing wasteful situations, the team can avoid expending more resources by going down the wrong path.

There are a number of ways that a team can achieve high-bandwidth communications. The Agile Manifesto (Beck et al. 2001) states that face-to-face communication is a preferred approach. Certainly, team members who are physically separated or primarily use noninteractive communication (such as documents) are at a disadvantage to colocated team members engaged in real-time, face-to-face collaboration.

Whenever possible, I prefer my team members to be colocated. However, many organizations, for various business reasons, have created distributed teams, so colocation may not always be possible or practical. I have worked with many distributed teams that have achieved the benefits of high-bandwidth communications, so being face-to-face isn't the only way to achieve the goal—but it's a great place to start if business conditions permit.

For distributed teams, a certain level of technology support can help improve communication bandwidth. I have worked with organizations where team members were widely distributed. Through the use of some rather impressive teleconferencing equipment, I participated in discussions that felt like everyone was colocated. Was it as good as being colocated? No. But the technology went a long way to improving the communication bandwidth among the team members.

Having teams composed of cross-functional team members is a critical step toward achieving high-bandwidth communications. Such teams have more streamlined communication channels simply because they have easy access to the people needed to get the job done. Also, such cross-functionally diverse teams are far less likely to have formal handoffs (which usually take the form of written documents) from one team to another. With everyone on the same team, the frequency and formality of handoffs are reduced, which improves communication speed.

We should also reduce time spent on ceremonies where team members perform a process that adds little or no value. For example, if team members have to go through three levels of indirection before they can speak with an actual customer or user, the ceremony of "talking to a customer" is probably a serious impediment to high-bandwidth communications. Having to create low- or no-value documents or requiring lengthy and potentially unnecessary approval and sign-off procedures reduces bandwidth. We need to identify and eliminate these impediments to improve overall team communication performance.

Finally, having small teams also improves bandwidth. Communication channels within a team do not scale linearly with the number of team members but instead increase by the square of the number of people on the team according to the formula $N(N-1)/2$. So, if there are 5 people on the team, there are 10 channels of communication. If there are 10 people on the team, there are 45 channels of communications. More people means more communication overhead and therefore lower bandwidth.

Transparent Communication

In addition to being high bandwidth (fast and efficient with minimal overhead), communication within the team should be transparent. Transparent communication

provides a clear understanding of what is actually happening to avoid surprises and help build trust among the team members. I have always felt that teams should communicate in a way that aligns with the spirit of the **principle of least astonishment**. Simply put, people should communicate in a way that is least likely to surprise one another. For example, I recall that on one Scrum team I coached, a particular individual would consistently choose his words during the daily scrums to occlude what he had accomplished and what he was planning to do. People were frequently surprised ("astonished") to later learn that his communications were intentionally opaque and designed to mislead. This resulted in other team members not trusting this individual, which in turn impeded the team's ability to self-organize and meet its sprint goals.

Right-Sized

Scrum favors small teams. The general rule is that having five to nine people on the team is best. There is published research that backs up the claim that small teams tend to be the most efficient (Putnam 1996; Putnam and Myers 1998). My experience over the past 25 years is that teams of five to seven are the sweet spot for rapidly delivering business value.

Mike Cohn lists a handful of reasons to keep teams small, which include the following (Cohn 2009):

- There is less social loafing—people exerting less effort because they believe that others will pick up the slack.
- Constructive interaction is more likely to occur on a small team.
- Less time is spent coordinating efforts.
- No one can fade into the background.
- Small teams are more satisfying to their members.
- Harmful overspecialization is less likely to occur.

It is possible to have too small a team. For example, a team is too small if it doesn't have the necessary people to get the job done, or if it has too few people to operate efficiently.

Just because Scrum favors small teams doesn't mean we can't use Scrum on larger development efforts. Scrum is frequently used to build products that require more than 9 people. However, rather than having one large Scrum team with, say, 36 development team members, we would instead have four or more Scrum teams, each with a development team of 9 or fewer people.

A Scrum project scales not by having a larger development team but by having multiple Scrum teams. Multiple Scrum teams can coordinate with each other in a variety of ways. One common approach is known as the **scrum of scrums**, where members of each Scrum team come together to perform a higher-level equivalent of the daily scrum (see Chapter 12 for details).

Focused and Committed

Team members need to be focused and committed to the team's goal. Focused means that each team member is engaged, concentrating on and devoting her attention to the team's goal. Committed means that during both good times and bad, each team member is dedicated to meeting the team's collective goal.

If a person is working on only one product, it is far easier for that person to be focused and committed. When asked to work on multiple concurrent product development efforts, a person must split her time across those products, reducing her focus and commitment on all products.

Ask any person who works on multiple products about her focus and commitment and you will likely be told something like "I have so much to do that I just try to do the best job that I can on each product and then hop to the next product. I don't ever feel like I have time to focus on any one product and do it well. If there is an emergency situation on several products, I simply won't be able to help out on all of them."

It is harder for a team member to do a good-quality job when she is hopping from product to product. It's even harder to be truly committed to multiple products simultaneously. Instead of being in one boat with her team members, the multitasking team member is moving from boat to boat. If many of the boats spring a leak at the same time, how does this person choose which boat's crew to help? If a person isn't there to bail water, that team member is not *committed* to that team. At best she is *involved* with that team. To be fair to the other team members the involved team member should make it perfectly clear that she is only involved and therefore might not be available at critical times.

There is considerable data to support the widely held belief that being on multiple products (or projects) or multiple teams reduces productivity. Figure 11.8 shows a graph of such data (Wheelwright and Clark 1992).

This data indicates that nobody is 100% productive—there is overhead just to be a good corporate citizen. Productivity actually seems better with two projects than with one. This occurs because it is possible to get blocked on one project, so having a second one to switch to allows a person to be incrementally more productive.

Based on this data, working on three or more concurrent projects is a bad economic choice because more time is spent on coordinating, remembering, and tracking down information and less time is spent doing value-adding work. So, how many projects/products (and probably different teams) should a person be on simultaneously? Probably not more than two. I have a strong preference for one, because in today's highly connected, information-rich world with email, instant messaging, Twitter, Facebook, and other technologies, being a good corporate citizen is probably the equivalent of being on one project!

Now what about those specialists who might need to be concurrently allocated to several products? Earlier I used the example of Sue (the UX designer), who was allocated 10% to a team (with the rest of her time going to other teams). As much as we

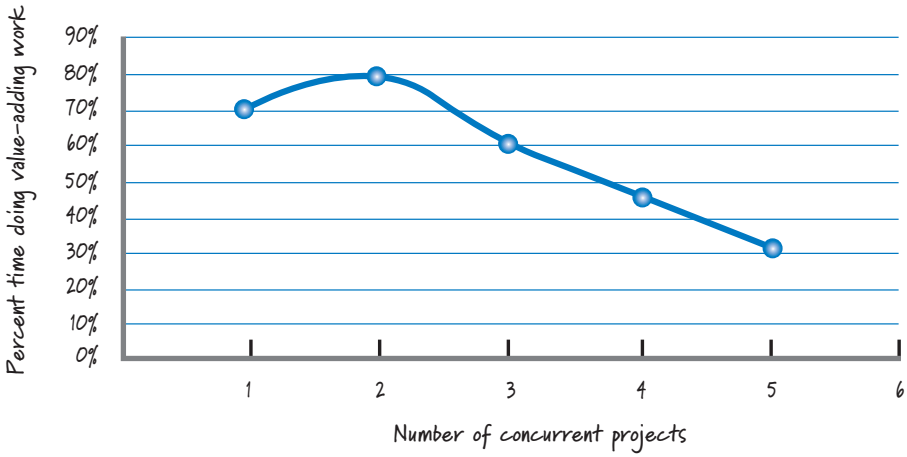

FIGURE 11.8 The cost of multitasking

would like Sue to be on one or two products, what if we need her part-time on five? As a practical approach, let the specialist decide how many products she can commit to and focus on simultaneously. If she says she can't commit to any more, don't assign her to that next product or team. If from a business perspective we are uncomfortable with her decision to not take on another product (let's say Sue is comfortable with only three concurrent products), perhaps we should seek an alternative solution to this problem.

Here are a few. First, do fewer projects concurrently. This is frequently the correct solution because many organizations have chosen to start too many projects at once (see Chapter 16 for a more detailed discussion). Another solution is to hire more specialists to share the burden. The third solution is to help other people broaden their skill sets to include the specialty skill. And, of course, the fourth solution is some combination of the first three solutions. In the end, forcing people to work on too many projects/teams concurrently will reduce their focus and commitment and jeopardize business outcomes.

Working at a Sustainable Pace

One of Scrum's guiding principles is that team members must work at a sustainable pace. (No more death marches!) In doing so, they deliver world-class products and maintain a healthy and fun environment.

Using sequential development, we defer important activities like integration and testing until near the end, when there typically is a crushing workload of issues to deal with as we approach the delivery date. The result is a steep increase in intensity in the latter phases (see Figure 11.9).

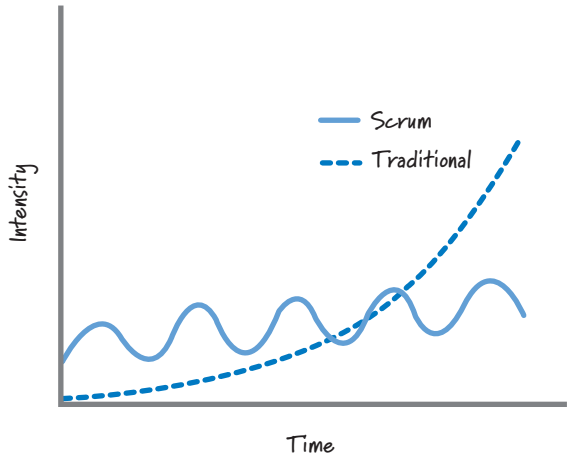

FIGURE 11.9 Sustainable pace over time

This incredibly intense time is symbolized by the superheroes pulling all-nighters and working weekends trying to get out the release. Some people thrive on this type of work, love the attention, and want to be rewarded for their extraordinary effort. The stress on everyone else is overwhelming. As an organization we should be asking, "Why did we have to work nights and weekends, and what should we change?"

Compare that with the typical intensity profile when using Scrum, where we've been continuously developing, testing, and integrating working features every sprint. During each sprint the team members should be using good technical practices such as refactoring, continuous integration, and automated tests to ensure that they can deliver value at frequent, regular intervals without killing themselves.

So, within a given sprint we'll likely see intensity increase a bit near the end of the sprint as we ensure that all work associated with our strong definition of done has been met. However, the overall intensity of work during each sprint should closely resemble the intensity of the previous sprint, reinforcing the team's working at a sustainable pace.

The aggregate result is a leveling of the work; it doesn't come in huge chunks or intense bursts, especially late when it is most harmful. This leveling means that Scrum teams will likely work fewer overtime hours and therefore be less likely to burn out.

Long-Lived

Effective use of Scrum requires teams, not groups. A **team** is made up of a diverse, cross-functional collection of collaborating people who are aligned to a common

vision and work together to achieve that vision. A **group** is a collection of people with a common label. Other than sharing the group name, group members don't share much else and won't effectively fulfill the responsibilities I described for the development team role.

As a rule, teams should be long-lived. I keep my teams together as long as it is economically sensible to do so. And the economics are very favorable for long-lived teams. Research by Katz has shown that long-lived teams are more productive than newly formed groups (Katz 1982). Furthermore, research by Staats demonstrates that team familiarity (team members' prior shared work experience) can positively impact the efficiency and quality of team output (Staats 2011). Improved productivity, efficiency, and quality lead to improved business results.

If we start out with a group of people who have never worked together, we have to spend time and money to get these people to gel into a real team. Most groups need to transition through phases, such as forming, storming, norming, and performing, to become highly functional teams (Tuckman 1965). Once we have a well-functioning team, we have a real business asset. Its members know how to work together, and they have earned each other's trust. In addition, the team has amassed important historical information, such as the team's velocity and shared estimating history (see Chapter 7). If we disband the team or significantly change its composition, this valuable, team-specific historical information no longer has a context for direct use.

Far too often I see organizations failing to appreciate the asset value of teams. Most organizations have developed skills and processes for moving people around to dynamically form "teams" (really groups). In my opinion such practices miss a critical aspect of Scrum—the value is in the team. The *currency of agile* is the team. In fact, one of the core values of the Agile Manifesto is "Individuals and Interactions." In other words, the team is the valuable asset.

Moving people around from team to team destroys the integrity of the team. I doubt that the New York City police SWAT (special weapons and tactics) team is recomposed with any frequency. Their team members have learned how to work together, and in a hot situation they have each other's backs. Moving people on and off that team would likely harm trust, integrity, and operational efficiency (a drop in velocity in our case, and in the specific case of the SWAT team, safety).

Most organizations would be far better off if they adopted a policy of keeping at least the core of their teams together as long as they can and moving teams from product to product. The economics of moving well-formed teams is almost always superior to the economics of moving people.

I'm not saying that you should always and can always keep your teams together for extended periods of time. For example, if we have a team that really hasn't gelled the way we had hoped, or is otherwise dysfunctional, it is often less disruptive and economically more sensible to disband the team.

In another case, I coached an organization where we knowingly broke up a high-performance Scrum team as part of a split-and-seed strategy for broadening the

adoption of Scrum within the organization. We didn't split the team because it completed its work and it was time to reassign people to new teams for the next development effort. Instead, we split it because we believed it was more valuable to form six new Scrum teams, each with a person experienced with Scrum, than to keep the original team together.

Finally, because teams are the assets, they are the unit of capacity that we should use to help establish the proper WIP limit on how many and which types of product development efforts we should pursue simultaneously. I will discuss this concept further in Chapter 16.

Closing

In this chapter I described the team role. I emphasized how the team is responsible for turning product backlog items into potentially shippable product increments. I also discussed the responsibilities of the team during each sprint. I then listed ten characteristics we want from our teams. In particular, we want team members who self-organize and are functionally diverse and sufficiently skilled to get the job done. Given the work the team must do, we want a good combination of T-shaped skills to enable effective swarming behavior. If people don't yet have the necessary breadth in their skills, we want people who are interested in acquiring that breadth.

We also want team members with an all-in-it-together Musketeer attitude. Teams should be created so that high-bandwidth communication is practical and encouraged. And we favor smaller rather than larger teams. To remain focused and committed, we prefer that team members work on only one or two development efforts at a time. Looking longer-term, we prefer to select team members who can stay together for an extended period of time on long-lived teams.

In the next chapter I will focus on the various Scrum team structures that you can use when scaling up your use of Scrum.